Axis2-based Replication Middleware for Web Services

Johannes Osrael[1], Lorenz Froihofer[1], Martin Weghofer[2], and Karl M. Goeschka[1]

[1]Vienna University of Technology, Argentinierstrasse 8/184-1, 1040 Vienna, Austria,
{johannes.osrael|lorenz.froihofer|karl.goeschka}@tuwien.ac.at
[2]University of Applied Sciences Technikum Wien, Höchstädtplatz 5, 1200 Vienna, Austria, weghofer@technikum-wien.at

Abstract

Dependability is one of the most important challenges for service-oriented architectures if their success shall continue in critical settings such as air traffic control or finance and banking. Replication of services and the underlying resources is one of the primary fault tolerance techniques for achieving dependability. While replication is well known in traditional fields (e.g. databases), it is rather in its infancy in service-oriented environments. Thus, in order to reduce the dependability gap we are currently facing in service-oriented environments, we contribute with a replication middleware for Web services which is built upon the Java-based Axis2 SOAP engine and provides a variant of primary-backup replication. Performance evaluations of our middleware implementation show the relatively low overhead of replication if the number of replicas is small.

1. Introduction

Replication is one of the primary means to achieve dependability [4], in particular high reliability and availability. As the success of service oriented architectures — and in particular Web services — continues and the service oriented approach is applied in high dependability demanding areas such as air traffic control or finance and banking, (Web) service replication is gaining importance as well.

Service oriented architectures can often be separated into a data layer (e.g. database back end) and a service layer (e.g. Web service front end) on top of it. In such a case, replication can be applied on the data layer and/or on the service layer [24]. On the data layer, traditional database replication techniques as provided by both commercial and open source database management systems (such as Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL, or MySQL) can be applied. Of course, only replicating the underlying data store is not sufficient — the Web service front end providing the business logic needs to be replicated as well. If the state of the Web service front end is completely encapsulated[1] in the underlying data store, "replication" of the front end becomes rather trivial since replica synchronization can be achieved via the replication mechanism of the data store and no coordination between the replicated front ends is required. If, however, the Web service front end contains some transient state or persists state in a data store not capable of replication, state synchronization needs to be performed on the service level.

Although obviously needed, only few replication middleware solutions for Web services have been presented in the past. Only some of them can be considered fault tolerant and state-of-the-art in terms of the used Web services technology, e.g. WS-Replication [29], which provides active replication [30]. Unfortunately, other solutions are not fully fault tolerant (e.g. FAWS [17]) or are already deprecated since they rely on outdated technology (e.g. FT-SOAP [20]). Hence, more research is required in this field.

Consequently, in this paper we contribute by presenting a [10] replication middleware for Web services built upon the Java-based Axis2 SOAP engine [28, 3].

The remainder of this paper is structured as follows: Our system model is presented in section 2. Afterwards, the system architecture is discussed in section 3. Results of the performance evaluation of the middleware solution are presented in section 4. Related work is discussed in section 5 before we draw our conclusion in section 6.

[1]This kind of services is also referred as "service that acts upon a stateful resource" [15]

2. System model

Our replication middleware is primarily targeted to replication of stateful Web services, since replicating stateless services is comparatively easy (w.r.t. stateful services) since no state needs to be synchronized. For stateless services, in principle, merely several instances need to be deployed on different hosts in the distributed system. Nevertheless, our middleware is beneficial for replication of stateless services as well, since it provides the abstraction of a logical service group (comprising several replicas), which reduces complexity for the application programmer.

2.1 Replication model

Our replication middleware has been designed for a variant of primary-backup replication, however, our modular design allows to plug-in other replication protocols (e.g. active replication [30] or coordinator-cohort replication [9]) with modest efforts.

In the original primary-backup approach [10], only one of the replicas — the primary — processes the clients' requests and forwards the updates to the other replicas — the backups. Thus, this technique is also called *passive* replication.

A variation of the original approach, which we have implemented in our middleware, is to forward the client invocations to the backups, which have to process them as well. Thus, the replica behavior must be deterministic in this variant. That is, sources for non-determinism such as non-deterministic functions (e.g., random(), time()) or the scheduling of concurrently executing conflicting transactions need to be avoided, e.g. by the use of a deterministic scheduler and by removing all non-deterministic function calls. Our primary-backup variant is similar to active replication [30] in the sense that all replicas are "active" and process the requests, but requires weaker multicast primitives (FIFO ordering instead of total order), since all client requests are forwarded to the primary replica first.

In primary-backup replication (including our variant), updates can be propagated either in a synchronous (eager, blocking) or asynchronous (lazy, non-blocking) fashion to the backup replicas. In the first variant, all replicas are updated before the reply is sent back to the client. Thus, replicas are always consistent and read operations can be performed on local copies. In the latter variant, the reply is sent to the client immediately after the primary has processed the request. Updates are propagated afterwards. This approach yields to better response time but replicas are not always consistent. Thus, read operations on backup copies might return stale values.

In a primary-backup approach, at least one replica (the primary) exists which has all updates. Moreover, FIFO ordering of operations is easy to achieve since all operations are directed to the primary. However, the primary replica might become overloaded. While a crash of a backup replica does not require specific actions by the replication protocol, a crash of the primary replica requires reconfiguration since a new primary needs to be elected.

Primary-backup replication can either be performed on the data level (i.e. via the underlying database) or on the service level of a stateful service. For a discussion on replication options in service-oriented systems see [24]. Our replication middleware performs replication on the service level and thus can be also used for services with transient state or data stores that are not capable of replication.

2.2 Failure model

We consider both node and link failures (partitioning), i.e. the crash failure [13] model is assumed for nodes and links may fail by losing but not duplicating or corrupting messages.

A group membership service is assumed in our system, which provides a single view of the nodes within a partition, i.e. it is used to detect node and link failures. Furthermore, we assume the presence of a group communication service which provides multicast to groups with configurable delivery and ordering guarantees.

2.3 Replication and service-orientation

Although one of the main goals of service-oriented architectures is inter-enterprise integration, replication as it is required and used in critical settings such as air traffic control is an intra-enterprise concern. That is, replication is not applied across organizational boundaries. Thus, replicas of a certain service can reside in a homogenous environment and replication middleware can be optimized for a certain kind of Web services technology, e.g., our replication middleware is targeted to Axis2 Web services. Therefore, today's replication frameworks for Web services can reuse many of the concepts of traditional (homogenous) replication frameworks used in distributed object systems or database systems as we have argued in previous work [25]. Some differences compared to traditional replication systems are caused by the coarse-grained nature of services, which allows certain optimizations (e.g., internal data structures) in the service replication middleware. Scalability, another major challenge addressed by service-oriented architectures, can be me mastered by looser forms of replication coupling. For instance, in case of our primary-backup scheme, loose replication coupling can be achieved by asynchronous update propagation.

While state-of-the-art Web service replication middleware frameworks (including our solution) address today's

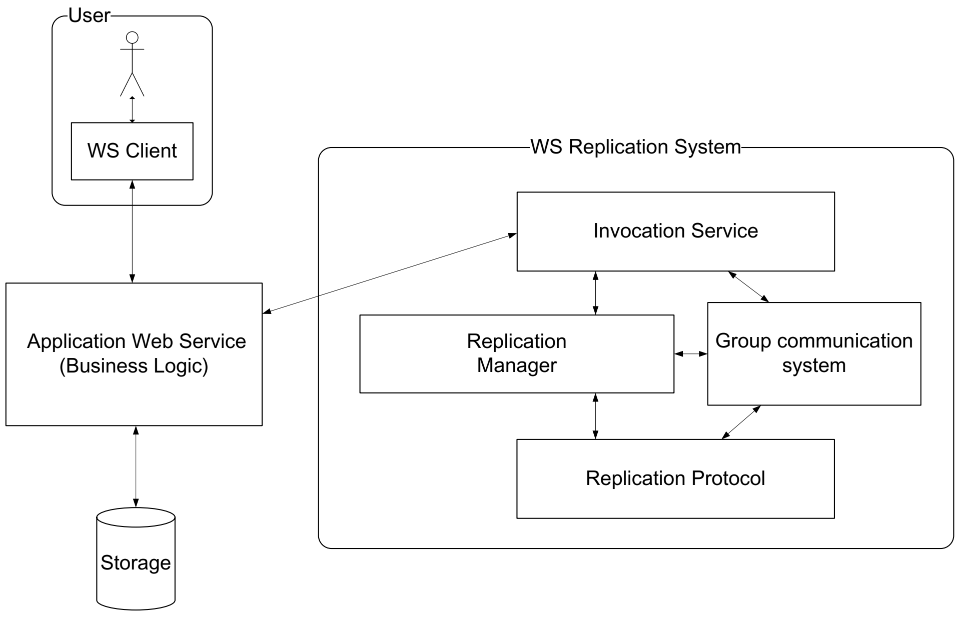

Figure 1. Middleware architecture

requirements for replication in service-oriented settings — which is still not trivial — we believe systems of the future such as ultra-large-scale systems [14] will require additional research if replication in a "true" service-oriented manner — especially with respect to heterogeneity — is required. Additional standardization efforts — similar to other horizontal protocols such as WS-Coordination [16] — for replication protocols, group membership services, group communication protocols, etc. are likely to be necessary for such settings [26].

3. System architecture

Our middleware consists of four main components as depicted in figure 1: an invocation service, a replication manager, a replication protocol, and the group communication toolkit Spread [2].

These middleware infrastructure components are implemented in a distributed fashion in order to avoid single points of failure and to provide fault tolerance for the middleware itself. That is, these components reside on every node in the system and their state (e.g. of the replication manager) is kept consistent.

Our replication middleware is built on the Java-based Apache Axis2 SOAP engine [28, 3] and realized as a module. Axis modules allow to extend the functionality of Axis in a flexible way. For instance, WS-Addressing [31], WS-Security [23], WS-ReliableMessaging [22] have also been implemented on top of Axis using the module concept.

The entry point to the replication middleware is the invocation service, which intercepts SOAP invocations of the client and triggers the replication logic.

3.1. Invocation service

Axis2 allows the definition of customizable message interceptors, so-called "handlers". Incoming SOAP messages are cut out of the Axis IN-flow, propagated to the other replicas and injected in their IN-flow. Each replica processes the SOAP invocation but a reply to the client is only sent by one of the replicas — the replica where the client invocation has been initiated.

The Axis flow is divided into phases, which are processed in sequential order. A handler is associated with each phase, i.e. first the handler of the TransportInPhase is called, afterwards the handler of the DispatchPhase, etc. The sequence of the phases is defined in the axis2.xml file. For replication purposes we have defined the phase "replicationPhase" (after the PostDispatchPhase) and an associated "InFlowReplicationHandler". The handler names and the associated handler classes and phases are defined in the module.xml file of the ReplicationModule.

Figure 2 shows the message flow in detail, assuming the client invocation has been initiated at the primary and synchronous update propagation is used. If the client invocation is initiated at a backup, it is redirected to the primary.

First, the message arrives at the HTTP-Transport interface (1) of Axis and is handed over to the handler chain. A handler receives the MessageContext via the invoke

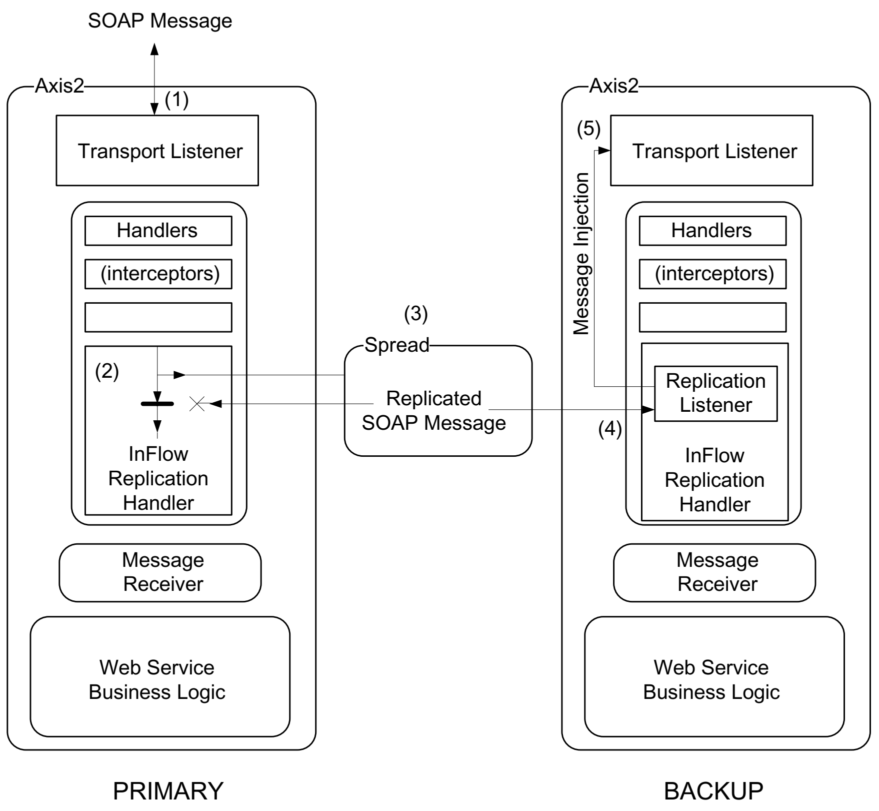

Figure 2. Replication handler and message injection

method. A MessageContext contains the Axis configuration, service and service group configuration, session information and the SOAP message. Since it is not possible to serialize the MessageContext in Axis2 and propagate it to other replicas, only the SOAP envelope and the To-endpoint are extracted by the InFlowReplicationHandler (2). The SOAP envelope is extended with a MessageHeader containing the MessageId and the host name of the primary. Afterwards, the SOAP envelope and the endpoint are stored in a ReplicationObject and multicast (3) to all replicas (including the primary) using the group communication toolkit Spread (see section 3.2 for group communication and section 3.4 for update propagation details). Sending the message to the primary as well is a convenient way to ensure consistency, since Spread delivers the message either to all or none of the group members. Otherwise state inconsistencies could occur, for instance if some replicas receive and process the SOAP message while others do not. The InFlowReplicationHandler at the primary is blocked till the SOAP message re-appears at the ReplicationHandler. On the receiver sites, the ReplicationObject is received by the ReplicationListener (4) and injected (5) in the Axis IN-flow using LocalTransportSender. Again, the message passes through the handler stack till it reaches the InFlowReplicationHandler. This handler checks if a SOAP header of the primary is contained in the SOAP envelope. If this is the case, the middleware knows that the received message is a replicated message and distinguishes two cases. If the receiver is the primary, the blocking of the original SOAP message is released, the original message is processed and the result is returned to the client. The replicated message is discarded. If the receiver is a backup, the replicated message is processed but no response is sent to the client.

3.2. Group communication toolkit

Reliable multicast primitives are required for propagation of messages from the primary to the backup replicas. Reliable multicast, however, requires concise information about which nodes are operational and which are not. Besides for reliable multicast, monitoring of the replicated Web services is also required to take appropriate action in case of a fault. For instance, a backup has to be promoted to the new primary if the original primary crashes.

Thus, agreement on the membership of nodes in the system has to be achieved. A group membership service keeps track of membership changes of *dynamic* groups, caused by voluntary (join or leave) changes or failures (crashed or un-

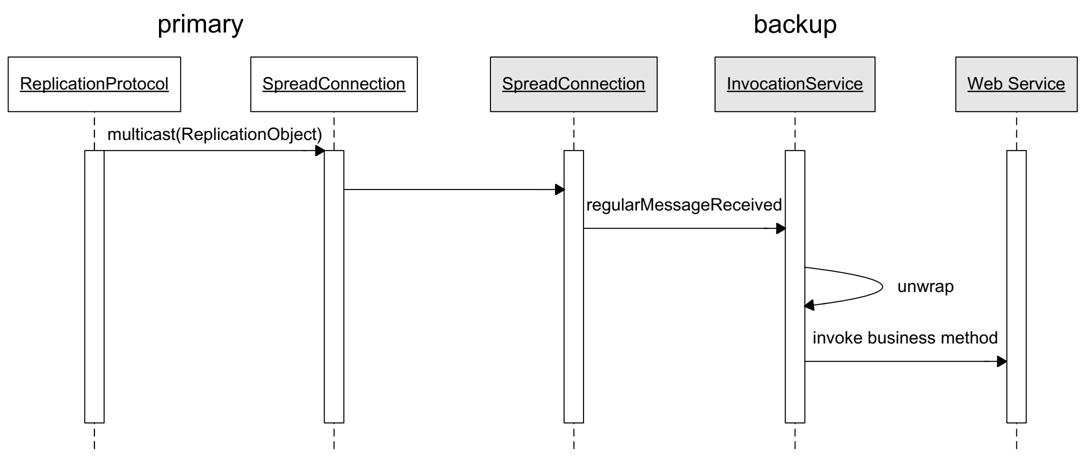

Figure 3. Update propagation

reachable nodes). A *view* contains the current members of a group. Group members are notified about group membership changes by the group membership service.

Since group membership changes have to be taken into account when a multicast is sent to a group, reliable multicast services are typically combined with a group membership service and referred as view-oriented group communication systems [12]. Group communication systems provide multicast primitives to (object, process, service) groups with configurable delivery and ordering guarantees.

Group communication systems have been widely used in the past as the basis for replication. For instance, group communication toolkits based on the virtual synchrony model [7, 8] developed at Cornell university by Birman's group "run the New York and Swiss stock exchange systems, the French air traffic control system, and the US Navy's Aegis-class warship" [6].

Since these toolkits hide much of the complexity for the programmer, we have chosen the Spread [2] toolkit for our replication middleware.

3.3. Replication manager

The purpose of the replication manager is to keep track of the location and roles of replicas, i.e. to manage the service groups. Thus, the replication manager processes membership messages[2] sent out by Spread and takes appropriate action if required. A membership message contains — among others — the members of the group. Since this list is in the same order on all nodes, a simple policy for determining the primary is to select the first node entry as the primary. In case of network partitions (indicated by a CausedByNetwork message), no new primary is selected and only the partition with the original primary can continue.

[2]Four different membership messages are distinguished: join, leave, disconnected, and network.

State synchronization Besides replication of SOAP invocations, the replication middleware also needs to provide a mechanism for state synchronization if new replicas of a service shall be added to the system or a crashed replica recovers. One way would be to store all SOAP invocations in a reliable data store and replay them once a new service replica shall be synchronized with the existing replicas. However, this can be quite time-consuming, especially if the number of SOAP messages is large. Thus, our replication manager foresees a state transfer mechanism. Services that shall be replicated must implement the StateTransfer interface which contains methods for getting and setting the state.

3.4. Replication protocol

The replication protocol realizes update propagation from the primary replica to the backup replicas. Update propagation is performed via the multicast primitives of the Spread group communication toolkit. We have defined a data structure called ReplicationObject which is a wrapper for the actual SOAP payload and contains some additional meta data such as the initiator of the message, a message number, etc. This ReplicationObject is serialized into a byte array and propagated via Spread.

Figure 3 schematically depicts the update propagation process (some implementation details are omitted).

4. Performance evaluation

We have evaluated the performance of our replication framework using a simple user account management Web

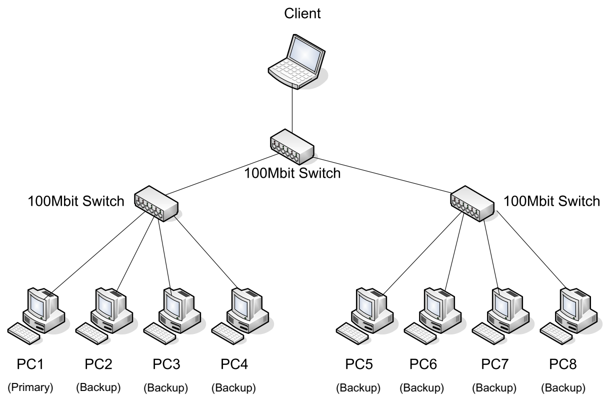

Figure 4. Test infrastructure

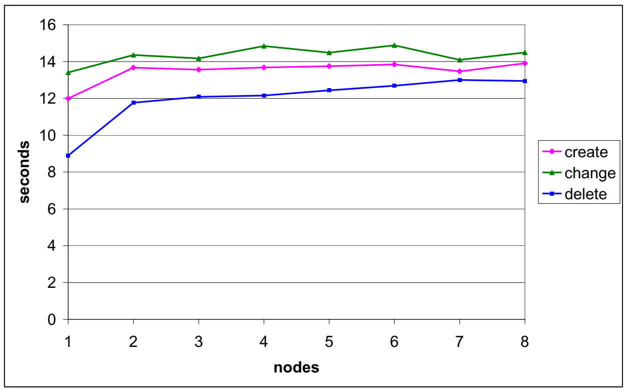

Figure 5. Update propagation: client invoked backup

service. The Web service provides four methods: login(), createUser(), changeUser(), and deleteUser(). The first method is read-only while the other methods have write semantics. The state of our Web services is stored in an xml file.

The Web service is replicated at up to eight nodes with the following characteristics: Fujitsu-Siemens Esprimo P5915 PCs, Intel Core 2 Duo E6600 processors, 2GB RAM, Windows XP Professional. The client resides on a laptop computer HP nx6110, Intel PentiumM 1,73GHz, 2GB RAM, Windows XP Professional. The Java Runtime 1.5.0_10, Axis2 1.1 and the HTTP server integrated in Axis are used on all nodes. The measurements have been performed in a 100Mbit local area network using three 100Mbit switches (Netgear Fast Ethernet Switch FS105 5Port). Figure 4 depicts the test infrastructure:

We measured the performance of our replication middleware for 1 to 8 nodes (i.e. with up to 7 backups). Four independent iterations of the following sequence have been performed: 400 x createUser(), 400 x login(), 400 x changeUser(), 400 x deleteUser(). Initially, the state of the Web service (i.e. the xml file) contained information about 203 users. Figure 5 shows the median of the duration for 400 invocations of each of the write methods, initiated at one of the backup replicas. The overhead of a replicated system compared to a non-replicated system (1 node) is (i) relatively small in a LAN setting and (ii) remains relatively constant for a small number of nodes. This result is in line with the performance measurements presented by Amir et al. [1] for the Spread toolkit. The scalability of our middleware primarily depends on the scalability of Spread, i.e. Spread's algorithms for group membership management and reliable multicast.

The performance of the login() operation is independent of the number of replicas, since synchronous update propagation is used by our replication protocol and thus read operations can be performed at any replica. 400 login() calls require approximately 7.5 seconds.

5. Related work

Few middleware solutions have been proposed for service-oriented systems:

Primary-backup replication of Web services is offered by the FT-SOAP (Fault-Tolerant SOAP) middleware [20]. While our replication middleware builds upon Axis2, FT-SOAP has been integrated with the predecessor Axis1. Due to a major redesign of Axis from version 1 to version 2, porting FT-SOAP to Axis2 would require significant effort. Moreover, many of the Web services standards have changed since the introduction of FT-SOAP in 2003.

FAWS [17] provides primary-backup replication as well but is not fault tolerant since the middleware components are not replicated.

A different failure model than ours—namely byzantine [19] behavior—is addressed by Thema [21]. Thema is based on the Castro-Liskov Practical Byzantine Fault Tolerance state machine replication protocol [11].

The WS-Replication [29] middleware offers transparent active replication, also based on Axis1. The major components in WS-Replication are a *Web service replication component* and a *reliable multicast component*. The former component enables active replication of Web services while the latter — called WS-Multicast — provides SOAP-based group communication. Moreover, WS-Multicast performs failure detection (which is required for group communication) by a SOAP-based ping mechanism. WS-Multicast can also be used independently from the overall WS-Replication framework for reliable multicast in a Web service environment. The SOAP group communication support has been built on the JGroups [18] toolkit. Currently, WS-Multicast is not available on an open-source basis, thus, we used the Spread group communication toolkit in our framework. Ye's and Shen's framework [32] also offers active replication based on group communication. Besides providing a different replication protocol compared to these frameworks, our solution is different to these approaches since it can be easily adapted for a variety of other protocols (e.g. coordinator-cohort) since replication management and protocol have been clearly separated in our framework.

ADAPT [5] is a J2EE replication framework integrated into the JBoss application server that allows to plug-in replication protocols. Besides replication of Enterprise JavaBeans it supports replication of Axis1 Web services as well. The Web services might contain session state but services that invoke other Enterprise JavaBeans or call a database are not supported.

As we have argued based on a comparison on an architectural level in [25] and discussed in an experience report [27], Web service replication middleware shares many commonalities with distributed object replication middleware.

6. Conclusion and future work

We have presented a primary-backup replication middleware for Web services built upon the Java-based Axis2 SOAP engine. The middleware has been implemented as an Axis2 module and constitutes four major architectural units:

- A *Replication Manager*, mainly for management of service groups and overall configuration of the replication logic.
- A *Replication Protocol* unit, primarily realizing update propagation.
- An *Invocation Service* for interception of client calls and triggering of the replication logic.
- The group communication toolkit *Spread* for reliable multicast to groups and membership monitoring.

Performance evaluations of our middleware implementation in a LAN setting show the relatively low overhead of replication if the number of replicas is small. While the middleware overhead for replication remains the same in a wide area setting, total response time from the client point of view will be higher due to increased network delays. If the performance overhead is not acceptable and replica consistency can be slightly weakened, asynchronous update propagation can be used instead of the synchronous variant chosen in our experiment.

As future work, we plan to implement other replication protocols such as the original primary-backup approach using state transfer for update propagation, active replication, and coordinator-cohort replication. Moreover, transactional support and a security concept shall be integrated.

7. Acknowledgements

This work has been partially funded by the European Community under the Framework Programme 6 IST project DeDiSys (Dependable Distributed Systems, contract number 004152, http://www.dedisys.org).

References

[1] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton. The spread toolkit: Architecture and performance. Technical report, Johns Hopkins University, 2004. http://www.cnds.jhu.edu/pub/papers/cnds-2004-1.pdf.

[2] Y. Amir, C. Danilov, and J. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proc. Int. Conf. on Dependable Systems and Networks*, pages 327–336. IEEE CS, 2000.

[3] Apache. Axis2, http://ws.apache.org/axis2/.

[4] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, 2004.

[5] O. Babaoglu, A. Bartoli, V. Maverick, S. Patarin, J. Vuckovic, and H. Wu. A framework for prototyping J2EE replication algorithms. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3291 of *LNCS*, pages 1413–1426. Springer, 2004.

[6] K. Birman. Can web services scale up? *IEEE Computer*, 38(10):107–110, 2005.

[7] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proc. 11th ACM Symposium on Operating systems principles*, pages 123–138. ACM Press, 1987.

[8] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1):47–76, 1987.

[9] K. Birman, T. Joseph, T. Raeuchle, and A. El Abbadi. Implementing fault-tolerant distributed objects.

IEEE Trans. on Software Engineering, 11(6):502–508, 1985.

[10] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. The primary-backup approach. In S.J. Mullender, editor, *Distributed systems*, chapter 8. ACM Press, Addison-Wesley, 2nd edition.

[11] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proc. of the 3rd Symp. on Operating Systems Design and Implementation*, pages 173–186. USENIX Association, 1999.

[12] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comp. Surveys*, 33(4):427–469, 2001.

[13] F. Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2), 1991.

[14] P. Feiler, R. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, L. Northrop, D. Schmidt, K. Sullivan, and K. Wallnau. *Ultra-Large-Scale Systems*. Software Engineering Institute Carnegie Mellon, 2006.

[15] I. Foster, J. Frey, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana. Modeling stateful resources with web services, 2004.

[16] IBM et al. Web services coordination, 2005. http://www-128.ibm.com/developerworks/library/specification/ws-tx/.

[17] D. Jayasinghe. FAWS for SOAP-based Web services, 2005. http://www-128.ibm.com/developerworks/webservices/library/ws-faws/.

[18] JGroups. JGroups: A toolkit for reliable multicast communication. http://www.jgroups.org.

[19] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[20] D. Liang, C.-L. Fang, C. Chen, and F. Lin. Fault tolerant web service. In *Proc. 10th Asia-Pacific Software Engineering Conf.*, pages 310–319. IEEE CS, 2003.

[21] M.G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service applications. In *Proc. 24th Symp. on Reliable Distributed Systems*, pages 131–142. IEEE CS, 2005.

[22] OASIS. WS Reliability 1.1, 2004. http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf.

[23] OASIS. Web Services Security - SOAP Message Security 1.1, 2006. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[24] J. Osrael, L. Froihofer, and K. M. Goeschka. *Software Engineering of Fault Tolerant Systems*, chapter Replication in Service-Oriented Systems. World Scientific Publishing, 2007.

[25] J. Osrael, L. Froihofer, and K.M. Goeschka. What service replication middleware can learn from object replication middleware. In *Proc. of the 1st Workshop on Middleware for Service Oriented Computing in conjunction with the ACM/IFIP/USENIX Middleware Conf. 2006*, pages 18–23. ACM Press, 2006.

[26] J. Osrael, L. Froihofer, and K.M. Goeschka. On the need for dependability research on service oriented systems. In *Proceedings of the 37th Int. Conference on Dependable Systems and Networks*. IEEE CS, 2007.

[27] Johannes Osrael, Lorenz Froihofer, and Karl M. Goeschka. Experiences from building object and service replication middleware. In *Workshop Proc. Int. Symposioum on Network Computing and Applications*. IEEE CS, 2007.

[28] S. Perera, C. Herath, J. Ekanayake, E. Chinthaka, A. Ranabahu, D. Jayasinghe, S. Weerawarana, and G. Daniels. Axis2, middleware for next generation web services. In *Proc. Int. Conf. on Web Services (ICWS'06)*, pages 833–840. IEEE CS, 2006.

[29] J. Salas, F. Perez-Sorrosal, Marta Patiño-Martínez, and R. Jiménez-Peris. WS-Replication: a framework for highly available web services. In *Proc. 15th Int. Conf. on World Wide Web*, pages 357–366. ACM Press, 2006.

[30] F.B. Schneider. Replication management using the state-machine approach. In S.J. Mullender, editor, *Distributed Systems*, chapter 2. ACM Press, Addison-Wesley, 2nd edition.

[31] W3C. Web Services Addressing 1.0 - Core, 2006. http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/.

[32] X. Ye and Y. Shen. A middleware for replicated web services. In *Proc. 3rd Int. Conf. on Web Services*, pages 631–638. IEEE CS, 2005.